

```

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-
web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-
test</artifactId>
            <scope>test</scope>
            <exclusions>
                <exclusion>
<groupId>org.junit.vintage</groupId>
                    <artifactId>junit-vintage-
engine</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>io.projectreactor</groupId>
            <artifactId>reactor-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

```

El siguiente paso es construir una primera versión del servicio REST de Hola Mundo en este caso utilizando la anotación `@RequestMapping`.

```
package com.arquitecturajava.router;
```

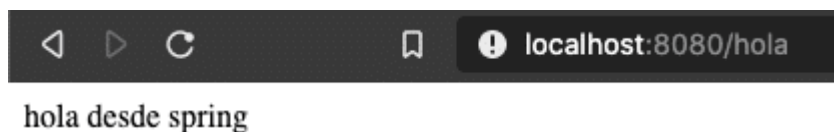
```
import org.springframework.web.bind.annotation.RequestMapping;
```

```
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HolaRESTService {

    @RequestMapping("/hola")
    public String hola() {
        return "hola desde spring";
    }
}
```

Si ejecutamos el proyecto de Spring boot



Nos mostrará los datos sin ningún problema.

Spring 4.3 GetMapping ,PostMapping etc

A partir de la versión 4.3 de Spring Framework existe la posibilidad de tener unas anotaciones algo más compactas y más sencillas de leer que se encarguen cada una de ellas de uno de los verbos HTTP (get,post, put,delete) . Veamos la modificación que es necesario aplicar.

```
package com.arquitecturajava.router;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

```
@RestController
public class HolaRESTService2 {

    @GetMapping("/hola2")
    public String hola() {
        return "hola desde spring con getmapping";
    }
}
```

El resultado es prácticamente el mismo solo que la anotación nos ayuda a simplificar y facilitar la lectura.



Spring WebFlux Router

A partir de Spring 5 tenemos la opción de usar Spring WebFlux Router que nos permite mapear el servicio sin añadir anotaciones al propio contenido de nuestra clase. Algo que permite centralizar y clarificar las configuraciones dejando las clases más limpias. Para que todo nos funcione correctamente deberemos actualizar el proyecto pom.xml para usar programación reactiva.

```
<dependencies>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-
webflux</artifactId>
    </dependency>
```

```

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-
test</artifactId>

            <scope>test</scope>
            <exclusions>
                <exclusion>
<groupId>org.junit.vintage</groupId>
                                <artifactId>junit-vintage-
engine</artifactId>
                                </exclusion>
                </exclusions>
            </dependency>
        <dependency>
            <groupId>io.projectreactor</groupId>
            <artifactId>reactor-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

```

Una vez realizada esta operación el siguiente paso es construir un servicio Reactivo **con Monos o Fluxs**.

```

package com.arquitecturajava.router;

import org.springframework.stereotype.Service;

import reactor.core.publisher.Mono;

@Service
public class HolaRESTService3 {

```

```

        public Mono<String> hola() {

            return Mono.just("hola desde spring y configuracion
funcional");
        }
    }
}

```

El último paso es cambiar el fichero de configuración de Spring para que acepte una nueva ruta.

```

package com.arquitecturajava.router;

import static
org.springframework.web.reactive.function.server.RequestPredicates.GET
;
import static
org.springframework.web.reactive.function.server.ServerResponse.ok;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import
org.springframework.web.reactive.function.server.RouterFunction;
import
org.springframework.web.reactive.function.server.RouterFunctions;
import
org.springframework.web.reactive.function.server.ServerResponse;
@SpringBootApplication
public class RouterApplication {

    public static void main(String[] args) {
        SpringApplication.run(RouterApplication.class, args);
    }
}

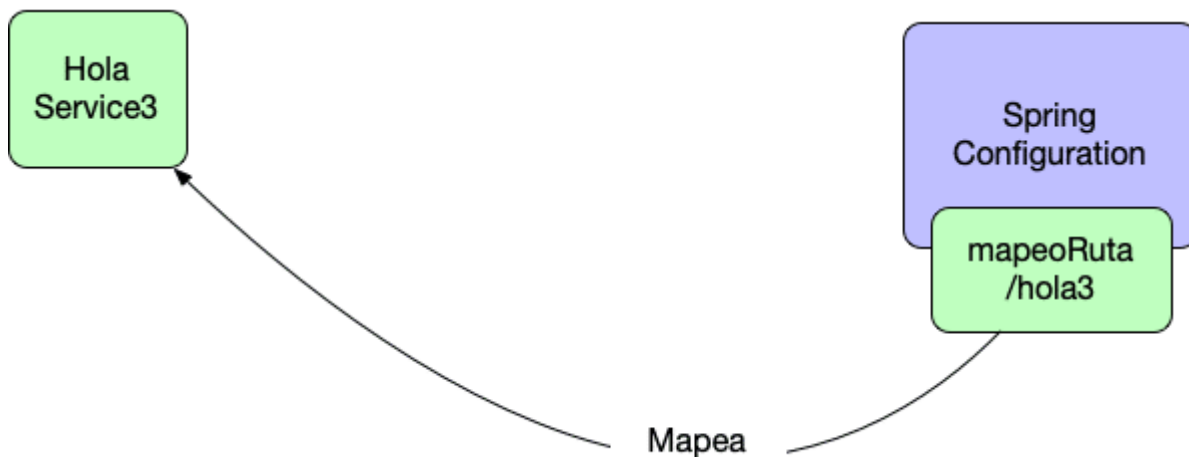
```

```

    }
    @Bean
    RouterFunction<ServerResponse> holaRoute(HolaRESTService3
holaService) {
        return
RouterFunctions.route(GET("/hola3"), req->ok().body(holaService.hola(),
String.class));
    }
}

```

Como podemos observar la ruta en este caso se denomina /hola3 y esta disponible a traves de un mapeo que se hace con programación funcional . El servicio no incluye ningún RequestMapping ni RestResource.



Si ejecutamos el programa el nuevo servicio queda publicado:



Conclusiones

Las nuevas capacidades de programación Reactiva de Spring 5 nos aportan nuevos enfoques en la programación y la programación funcional suma a la hora de abordar configuraciones compleja.

- [Spring Boot](#)
- [Spring WebFlux Test](#)
- [Spring Initializer](#)
- [Spring WebFlux](#)